**Recommendation Systems**

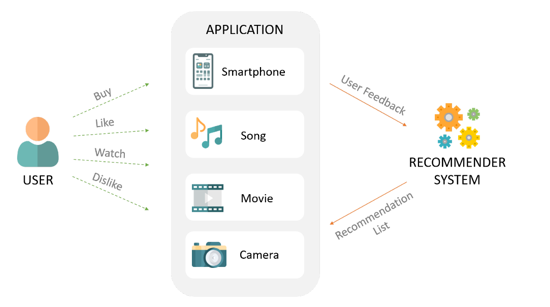
Recommendation systems, the terms itself seem to be self-explanatory, a system or an automated system that provides suggestions/recommendations.

For example, when you watch a movie of some particular genre on Netflix or watch some video on YouTube, they start to give you suggestions/recommendations for similar content types.

Other examples can be, when you do shopping on amazon, after viewing some product, it starts suggesting a similar product or listening to music on some music app, like Spotify. It starts to suggesting songs you may also like to listen to.

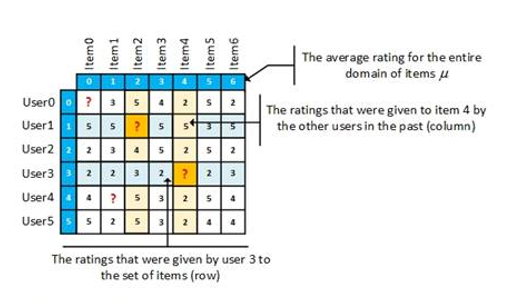
There can be several examples where recommendation systems are being used today, but if you notice, they all carry a similar type of trait, i.e. using the customer data to provide recommendations.

Therefore, we can say that recommendation systems use customer data (what you watch, what you buy what music you listen to, etc.) and find patterns in that data and on the basis of these patterns provide personalized suggestions to improve customer experience.



**Data representation for Recommendation systems**

The customer data used to build Recommendation systems is represented as a matrix of users and items. This matrix can vary according to the problem at hand. For eg. See the figure below, represents the matrix where rows are users and columns are movies, and the value in the matrix represents the rating given by a particular user to some movie.



This Matrix is also called a user-item interaction matrix. ‘?’ in the matrix represents that the user has not rated that movie and there is no interaction between them.

**In reality, most of the data points in a matrix are ‘?’ there are a very large number of users and a large number of movies, and only a few users rate few movies.**

Like every machine learning algorithm, a recommender system makes predictions based on users' historical behaviors. Specifically, it's to predict user preference for a set of items based on past experience. To achieve this task, there exist many methods.

**Averaging**

One simple way of doing this is, we assume that every user is the same and we predict values by the average of the column, or in our case prediction is the average rating given to some movie.

**Content-based filtering**

Content-based approaches use additional information about users and/or items. If we consider the example of a movies recommender system, this additional information can be, for example, the age, the sex, the job or any other personal information for users as well as the category, the main actors, the duration, or other characteristics for the movies (items).

Then, the idea of content-based methods is to try to build a model, based on the available “features”, that explain the observed user-item interactions. Still considering users and movies, we will try, for example, to model the fact that young women tend to rate better some movies, that young men tend to rate better some other movies, and so on. If we manage to get such a model, then, making new predictions for a user is pretty easy: we just need to look at the profile (age, sex, …) of this user and, based on this information, to determine relevant movies to suggest.

**Collaborative Filtering**

More Methods for Prediction in Recommendation systems.

The first method we saw in the last module was to predict using the average of all the items assuming all the users are the same. But this method seems to be very naïve, as it carries an assumption of all users being the same. One of the solutions for this problem is **clustering-based recommendation systems.**

**Clustering:**

We have learned about clustering in the previous weeks and how it can be used to find groups of similar types of data. In recommendation systems we make clusters based on users, item interactions and each cluster would be assigned to typical preferences, based on the preferences of customers who belong to the cluster. Customers within each cluster would receive recommendations computed at the cluster level.

**Clustering is a bit weak** because what we do in fact is identify user groups and recommend each user in this group the same items.

**Collaborative filtering**

Collaborative filtering is one of the most popular approaches used in recommendation systems as it helps in providing personalized recommendations based on similar users having similar interests. It looks at the items they like and combines them to create a ranked list of suggestions. There are many ways to decide which users are similar and combine their choices to create a list of recommendations.

To find similarity between different users, We can use either distance-based similarity measures like euclidean, manhattan, etc or we can find cosine similarity between different users.

After you have determined a list of users similar to a user **U**, you need to calculate the rating **R** that **U** would give to a certain item **I**. Again, just like similarity, you can do this in multiple ways.

You can predict that a user’s rating **R** for an item **I** will be close to the average of the ratings given to **I** by the top 5 or top 10 users most similar to **U**.

Another way is by taking the weighted average of different users.

Now, you know how to find similar users and how to calculate ratings based on their ratings. There’s also a variation of collaborative filtering where you predict ratings by finding items similar to each other instead of users and calculating the ratings.

The technique in the examples explained above, where the rating matrix is used to find similar users based on the ratings they give, is called user-based or user-user collaborative filtering. If you use the rating matrix to find similar items based on the ratings given to them by users, then the approach is called item-based or item-item collaborative filtering.

The two approaches are mathematically quite similar, but there is a conceptual difference between the two. Here’s how the two compare:

* **User-based:** For a user **U**, with a set of similar users determined based on rating vectors consisting of given item ratings, the rating for an item **I**, which hasn’t been rated, is found by picking out N users from the similarity list who have rated the item **I** and calculating the rating based on these N ratings.
* **Item-based:** For an item **I**, with a set of similar items determined based on rating vectors consisting of received user ratings, the rating by a user **U**, who hasn’t rated it, is found by picking out N items from the similarity list that have been rated by **U** and calculating the rating based on these N ratings.

Singular Value Decomposition/Thresholding:

In reality, there are some latent features that explain the behavior of users and items. Let's suppose there some latent features U and V for the users and items, as usually, we have seen till now, the features have many missing values, so the purpose of SVD is to estimate the matrix by filling null values with 0 and applying the following formula to estimate the feature matrix.

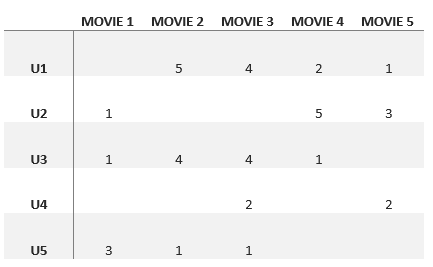
**A= USVT**

**Matrix A is the estimation of the feature matrix.**

**Matrix Factorization**

Matrix factorization is a method to create latent features when multiplying two different kinds of entities(user, movies). Collaborative filtering is the application of matrix factorization to identify the relationship between items and users entities. With the input of users ratings on the items, we would like to predict how the users would rate the items so the users can get the recommendation based on the prediction.

Assume we have the customers’ rating table of 5 users and 5 movies, the matrix is provided by the table below.

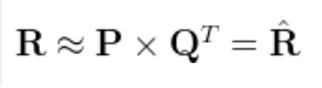


As not all movies are being rated by every user, there are many missing values in the matrix. The missing values could be filled with 0 such that the filled values are provided for the multiplication. For example, two users give high ratings to a certain movie when the movie is acted by their favorite actor and actress or the movie genre is an action one, etc. From the table above, we can find that user1 and user3 both give high ratings to move2 and movie3. Hence, from the matrix factorization, we are able to discover these latent features to give a prediction on a rating with respect to the similarity in user’s preferences and interactions.

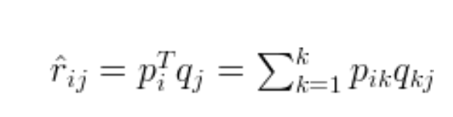
Given a scenario, user 4 didn’t give a rating to movie 4. We’d like to know if user 4 would like movie 4. The method is to discover other users with similar preferences of user 4 by taking the ratings given by users of similar preferences to the movie 4 and predict whether user 4 would like the movie 4 or not.

Mathematic concept of matrix factorization

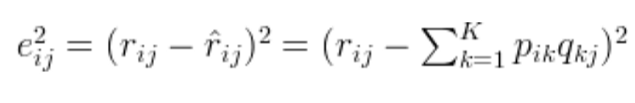
Define a set of Users U, items D, R size of matrix U, and D. The matrix U\*D includes all the ratings given by users. The goal is to discover**K**latent features. Given the input of two matrics matrices P (|U|\*k) and Q (|D|\*k), it would generate the product result R.



Matrix P represents the association between a user and the features while matrix Q represents the association between an item and the features. We can get the prediction of a rating of an item by the calculation of the dot product of the two vectors corresponding to u\_i and d\_j.



To get two entities of both P and Q, we need to initialize the two matrices and calculate the difference of the product named matrix M. Next, the total error E is minimized using gradient descent, aiming at finding a local minimum of the difference.





**Markov Chain**

What is a Markov Chain?

Google is the most preferred one-stop destination for all online searches, known for its simple, quick, and useful results. Having started off as a search engine, Google ranks pages as output and accords an order to the web world. Did you know, the common interpretation of the PageRank algorithm relies on the simple mathematical notion of Markov chains?

Markov chains are a fundamental part of stochastic (random) processes, widely used across disciplines. The process satisfies the Markov property, implying that the past and future are independent when the present is known. Simply put, if one knows the current state, then no additional information of past states is required to make the best possible prediction of the future.

The Markov property makes it much easier to study the random process. If we know the value taken by the process at a given time, we will not get any additional information about the future behavior by gathering more inputs about the past. In slightly more mathematical terms, for any given time, the conditional distribution of future states of the process, given the present and past states, depends only on the present state and not at all on the past states (memoryless property). A random process with the Markov property is called the **Markov process.**

For example, if you made a Markov chain model of a baby's behavior, you might include playing, eating, sleeping, and crying as states, which coupled with other behaviors, could form a 'state space': a list of all possible states. In addition, on top of the state space, a Markov chain tells you the probability of hopping or transitioning from one state to any other ---e.g., chance that a baby which is currently playing may fall asleep in the next five minutes without crying first.

Markov chains are used to compute probabilities of events occurring, by viewing them as states transitioning into other states, or transitioning into the same state as before. A Markov chain is represented by using a probabilistic automaton. The changes of state of the system are called transitions. The probabilities associated with various state changes are called transition probabilities. A probabilistic automaton includes the probability of a given transition into the transition function, turning it into a transition matrix.

When approaching Markov chains there are two different types; discrete-time Markov chains and continuous-time Markov chains. This means that we have one case where the changes happen at specific states and one where the changes are continuous, we shall learn about its states and shall also focus on some important properties associated with Markov Chains like,

**Reducibility:** A Markov chain is irreducible if it is possible to get to any state from any state, or if there is a chain of steps between any two states that have a positive probability.

**Periodicity:** A state in a Markov chain is periodic if the chain can return to the state only at multiples of some integer larger than 1. Thus, starting in state 'i', the chain can return to 'i' only at multiples of the period 'k', and k is the largest such integer. State 'i' is aperiodic if k = 1 and periodic if k > 1.

**Transience and Recurrence:** A state 'i' is said to be transient if, given that we start in state 'i', there is a non-zero probability that we will never return to 'i'. State i is recurrent (or persistent) if it is not transient. A recurrent state is known as positive recurrent if it is expected to return within a finite number of steps and null recurrent otherwise.

**Ergodicity:** A state 'i' is said to be ergodic if it is aperiodic and positive recurrent. If all states in an irreducible Markov chain are ergodic, then the chain is said to be ergodic.

**Absorbing State:** A state ‘i' is called absorbing if it is impossible to leave this state. Therefore, the state 'i' is absorbing if pii = 1 and pij = 0 for i ≠ j. If every state can reach an absorbing state, then the Markov chain is an absorbing chain. To put simply, Markov properties makes the study of these processes much more tractable and allows to derive some interesting explicit results.

To put it simply, Markov properties make the study of these processes much more tractable and allows to derive some interesting explicit results.

Since Markov chains can be designed to model many real-world processes, they are used in various situations. These fields range from the mapping of animal life populations to search-engine algorithms, music composition, and speech recognition. In economics and finance, they are often used to predict macroeconomic situations such as market crashes and cycles between recession and expansion. Other areas of application include predicting asset and option prices and calculating credit risk. When considering a continuous-time financial market, Markov chains are used to model the randomness.

In short, after completing this module,

1. You can track the different touchpoints the customer has encountered before making the final purchase; and
2. You can learn about the role of each channel in the customer journey, prioritize the channel and see how many conversions are happening without the channel being in place, using the Removal effect principle.

**Hidden Markov Model (HMM)**

Let’s begin with the Markov Chain. A Markov Chain or Markov Model is a special type of the discrete stochastic process in the probability theory. Here the probability of an event occurring only depends on the immediately preceding event, based on an underlying assumption that the “future is independent of the past, given the present”. In other words, if we know the present state or value of a system or variable, we do not need any past information to try to predict the future states or values.

Markov chains are generally defined by a set of states and the transition probabilities between each state. These probabilities are usually represented in the form of a Transition Matrix or the Markov Matrix.

**Hidden Markov Models** are probabilistic models that attempt to find the value or probability of certain hidden variables, based on other observed variables. These variables are commonly referred to as hidden states and observed states.

The state of a system could either be partially observable or not observable at all. Hence, we may even have to infer its characteristics, based on another fully observable system or variable. For example, you might have the full expertise to determine the returns on investment but cannot arrive at a conclusive figure without certain information (missing pieces).

In practical terms, assume you have the value of returns of all the assets in your portfolio; yet without knowing the rate at which each asset produces the returns, we cannot have a true reflection of the portfolio returns at a specific point in time, and cannot provide an accurate returns estimate.” The process termed the Hidden Markov model may be used to shed some light on this problem.

It is split between the observable component and the unobservable or ‘hidden’ component. Nevertheless, the observable process can help us extract information about the “hidden” processes. As such our task is to determine the unobserved process from the observed one.

The Hidden Markov Models (HMM) has two defining assumptions.

1. Observation at a specific point of time was generated by some process whose state is hidden from the observer; and
2. State of this hidden process satisfies the Markov property

During the course, we shall also learn about some properties and definitions that will strengthen our grasp of the HMM concept.

* **Time homogeneity:** This occurs when the probability of moving from point ‘a’ to ‘b’ is independent of time, that is, it does not matter how far you are in the process; as long as the processes are going to move from ‘a’ to ‘b’ in one step, the probability will be the same throughout. When a process has this property, we say it is time homogenous and if not; time non-homogenous.
* **Irreducible states:** It is possible to move from any one state to another over a certain number of steps.

We all know that the probabilistic graphical models are classified as directed graphical (Bayesian Networks) and undirected graphical models (Markov Random Fields or MRFs). Directed graphical models can be represented by a graph with its nodes serving as random variables and directed edges serving as dependency relationships between them. In undirected graphical models, dependencies between nodes are similar to the Bayesian network, but connections between undirected edges represent joint probabilities.

The fundamental property of MRFsis to satisfy the pairwise local and global Markov properties. These properties share a deep connection with the d-separation rules for Bayesian networks. In particular, d-separation defines the Markov properties on directed graphs.

Further, we shall find out the ways to extract messages passing on the junction trees. Within each network, there are multiple nodes, which are interlinked to form trees. Cliques are the junctions between two trees. Directions of these trees are specified by messages. To begin with, we shall learn to construct a clique tree from a graph. A clique tree is a very versatile data structure. Factors can be computed in the cliques and messages are sent along edges, eliminating the unnecessary variables. To infer from the graphical models, we use algorithms such as junction tree, sum-product belief propagation, elimination algorithms, etc.

Till this point, we have assumed that model parameters and graph structures are known. But in many applications, both are unknown and need to be estimated basis the available data. In such cases, we shall use parametric estimates and apply algorithms such as maximum likelihood, proportional iterative filtering (IPF), and expectation-maximization.

Hidden Markov models are applied in a host of fields including thermodynamics, statistical mechanics, physics, chemistry, economics, finance, signal processing, information theory, pattern recognition -such as speech, handwriting, gesture recognition, and bioinformatics.

Key takeaways from this module:

1. What is a Hidden Markov Model and how to find the most likely sequence of events, using a collection of outcomes and limited information?
2. How to structure a computation so as to maximize efficiency and links between computational complexity and graph structure?
3. What phenomena can be captured by different classes of graphical models and the links between graph structures, and how to decipher its representational power?